Федеральное государственное автономное образовательное учреждение высшего образования   
«Пермский национальный исследовательский политехнический университет»

«Задача о восьми ферзях»

Выполнил студент группы ИВТ-23-2Б

Муравьев Дмитрий Александрович  
Проверила: доцент кафедры ИТАС

Ольга Андреева Полякова

2023

1. Постановка задачи:

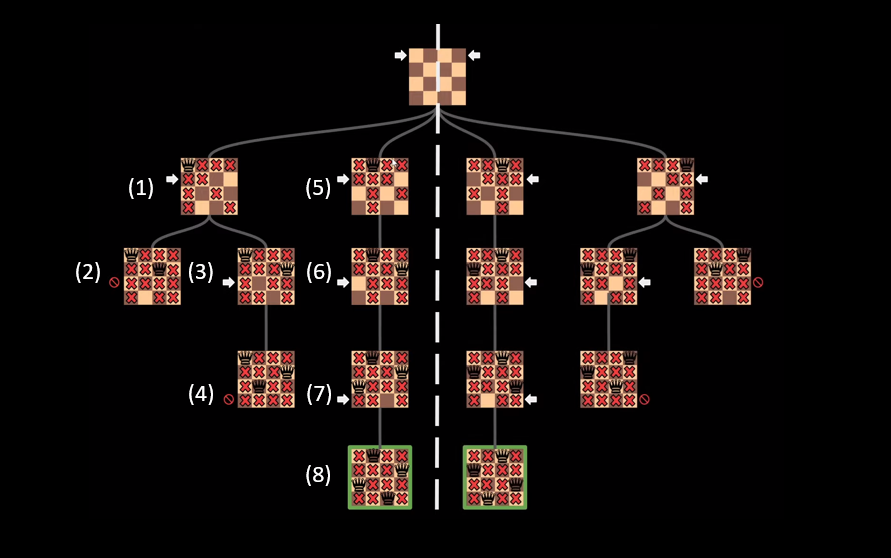
Расставить на стандартной 64-клеточной шахматной доске 8 ферзей так, чтобы ни один из них не находился под боем другого.

В своей работе я выполнил как поиск одного любого решения, так и всех решений для n ферзей на доске n на n.

2. Анализ задачи:

Для решения задачи будем использовать алгоритм поиска с возвратом. В английской литературе этот метод называется backtracking.

Для лучше понимания сути алгоритма я привел изображение, на которое буду ссылаться во время объяснения.

****

Шаг 1: ставим первого ферзя в первую клетку первого ряда (левый верхний угол).

Шаг 2: ставим второго ферзя в первую клетку, находящуюся не под боем следующей строки.

Шаг 3: попытаюсь сделать шаг 3 аналогично предыдущим, мы обнаружим, что на следующей строке мы не можем поставить ферзя. Поэтому вместо того, чтобы ставить следующего ферзя, мы переставим последнего поставленного ферзя на следующую клетку в строке.

Шаг 4: ставим третьего ферзя в первую клетку следующей строки, находящуюся не под боем.

Шаг 5: мы не можем поставить последнего ферзя, так как все клетки под боем. Мы так же не можем переставить последний и предпоследний поставленные ферзи. Поэтому мы их убираем с доски и переставляем первого ферзя.

Шаг 6: ставим второго ферзя в первую клетку следующей строки, находящуюся не под боем.

Шаг 7: ставим третьего ферзя в первую клетку следующей строки, находящуюся не под боем.

Шаг 8: ставим четвертого ферзя в первую клетку следующей строки, находящуюся не под боем.

Задание решено, мы расставили всех четырех ферзей так, что не один из них не находится под боем.

На картинке видно, что все иные постановки первого ферзя приведут к отзеркаленным версиям уже разобранным ранее позициям.

Анализируя данный алгоритм, мы можем заметить, что нам необходимо несколько раз повторять одни и те же действия (ставим ферзя, пробуем поставить следующего, если не можем, переставляем предыдущего и т.д.). Поэтому при реализации алгоритма на языке программирования будет разумно использовать рекурсивную функцию, которая будет вызывать себя и тем самым повторять необходимые нам действия.

Для хранения и обработки расстановки фигур на шахматной доске я буду использовать двумерный массив целых чисел, вложенные массивы которого будут строками шахматной доски. Числа 0 будут пустыми клетками, не находящимися под ударами, числа -1 – пустые клетки, находящиеся под боем, а числа 1 – клетки, на которых стоят ферзи.

Для вывода расстановки фигур на шахматной доске я буду использовать функцию print\_board. Так же внутри этой функции я буду указывать за сколько шагов мы дошли до этой расстановки.

Внутри рекурсивной функции для проверки находится ли эта клетка под ударом, перед тем как поставить на нее ферзя, я буду вызывать функцию check\_safety. Она будет принимать в качестве аргументов координаты клетки и возвращать true, если поле безопасно, и false, если оно уже под боем. Для того что бы определить находится ли клетка под ударом, я буду циклически проверять есть ли в этой строке, столбце или диагонали другие ферзи. Если хоть один ферзь найдется функция закончит проверку и вернет false. Иначе функция вернет true.

После постановки или снятия с доски каждого ферзя я буду вызывать функция mark\_under\_attack. В качестве аргументов функция будет принимать координаты ферзя и значение, на которое нужно поменять элементы массива. Таким образом после постановки ферзя, в функцию вместе с координатами я буду передавать значение -1, чтобы отмечать поля, как находящиеся под боем. Если же я убираю ферзя с доски, то в функцию я буду передавать значение 0, чтобы отмечать поля, как не находящиеся под боем. Внутри самой функции я буду циклически проходить по строке, столбцу и диагонали и, если будут найдены поля, еще не находящиеся под боем других ферзей, то их значения будут заменятся на -1 или 0.

Для организации рекурсивного поиска решения я буду использовать функцию solve\_searching. В качестве аргументов функция будет принимать столбец, в который сейчас надо ставить ферзя. Функция будет возвращать true и false, если решение есть или его нет соответственно. Если текущий столбец равен общему количеству столбцов (программа работает с нулевого столбца), то это значит, что все ферзи расставлены и задача решена. Иначе мы проверяем по очереди все клетки столба, можем ли мы поставить на какую ни будь из них ферзя, вызывая с координатами интересующей нас клетки в качестве аргументов функцию check\_safety. Если можем, то ставим ферзя и отмечаем клетки, которые теперь под ударом (функция mark\_under\_attack с значением -1). Затем функция взывает себя с следующим столбцом в качестве аргумента и проверяет есть решения, идущие от текущей расстановки фигур, а доске, если решений нет, то мы убираем последнего поставленного ферзя, и с сбрасываем пометку клеток (функция mark\_under\_attack с значением 0).

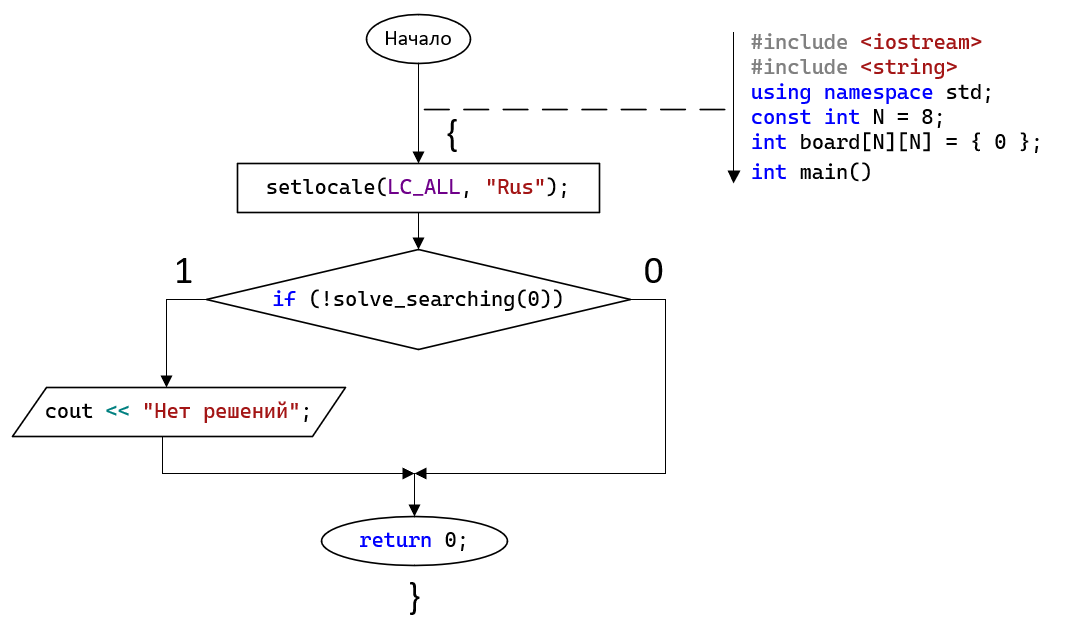
В функции main мы только вызываем функцию solve\_searching, одновременно проверяя есть ли решения. (Для 2 и 3 ферзей на доске 2 на 2 и 3 на 3 решений нет)

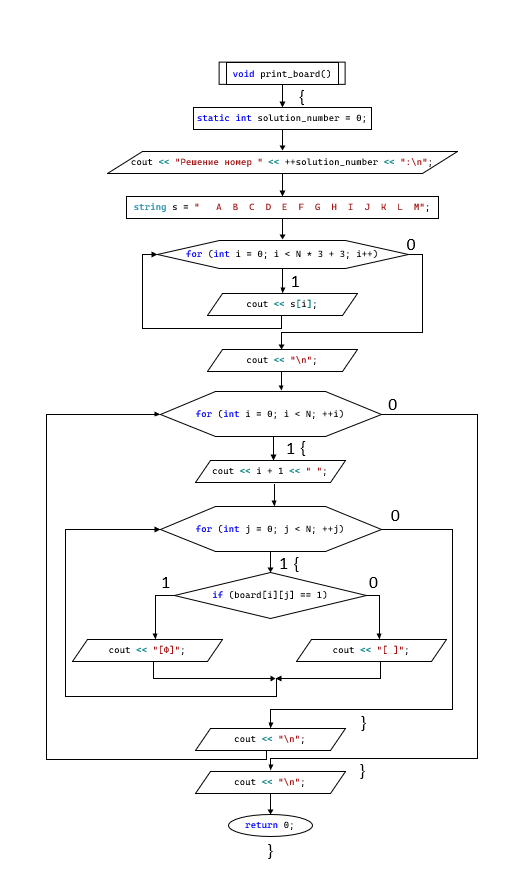
У меня получилась программы, которая выводит каждый шаг программы на шахматной доске. Моя программа получилась универсальной, если изменить константное значение переменной N, которая определяется и инициализируется в пятой строке кода, можно получить решение для любого числа ферязей на соответствующим им полям. Однако на решение задачи для 11 и более ферзей требуется много времени.

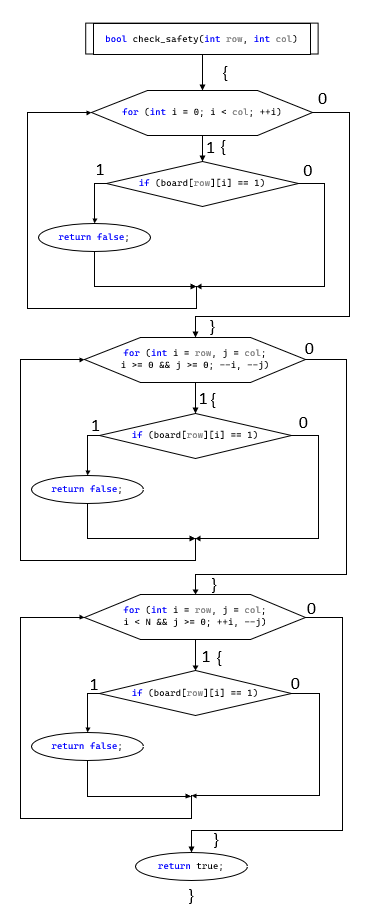
Также, слегка изменив программу, я обеспечил нахождение всех возможных решений для этого числа ферзей их подсчета и вывода в консоль.

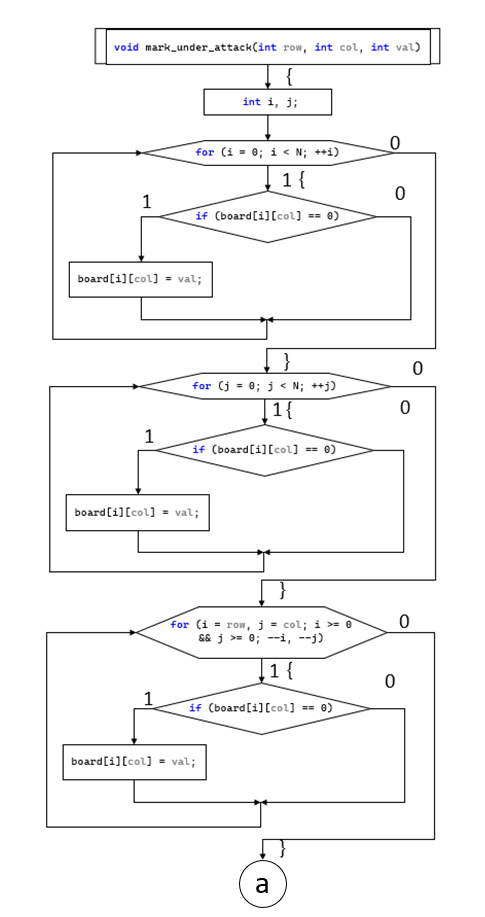
3. Блок схема

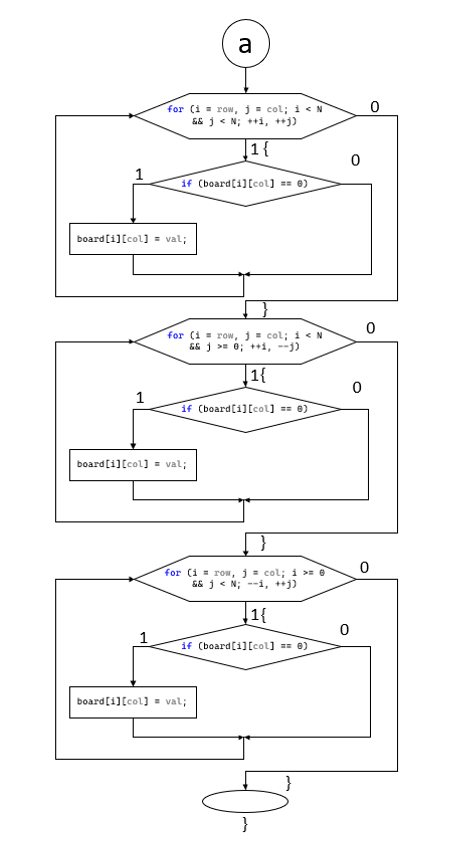
Блок схема программы для поиска одного любого решения:

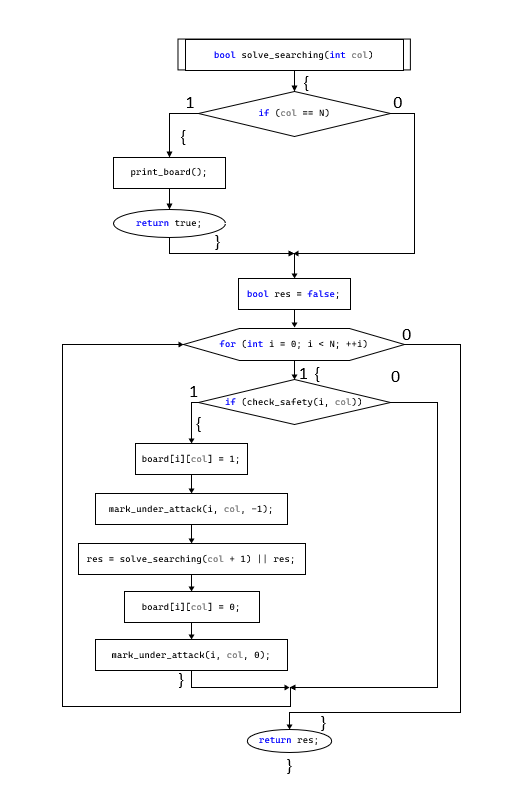




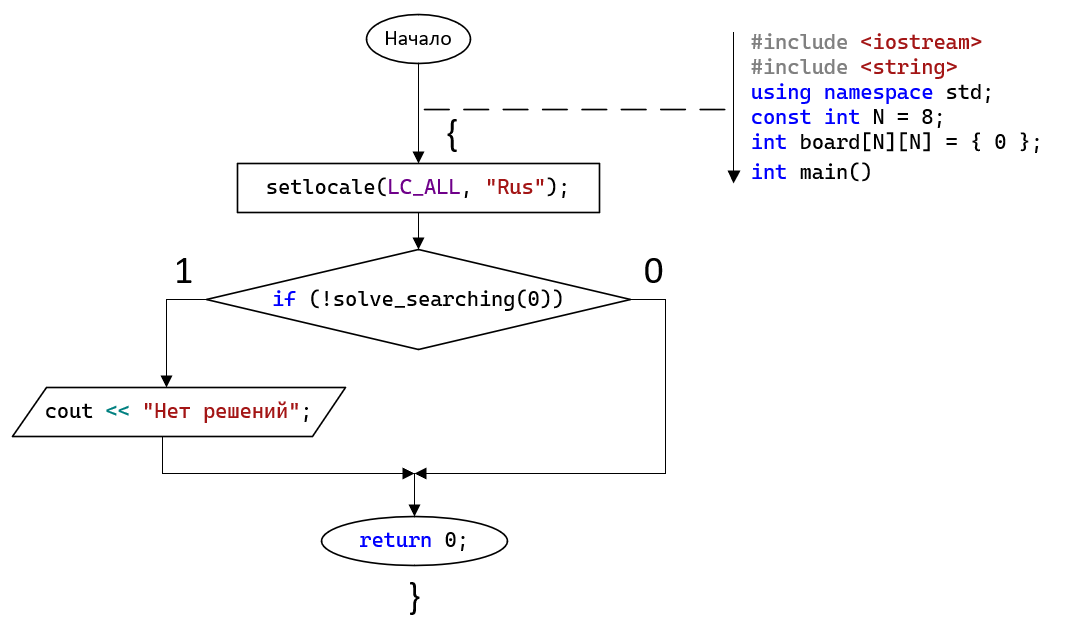


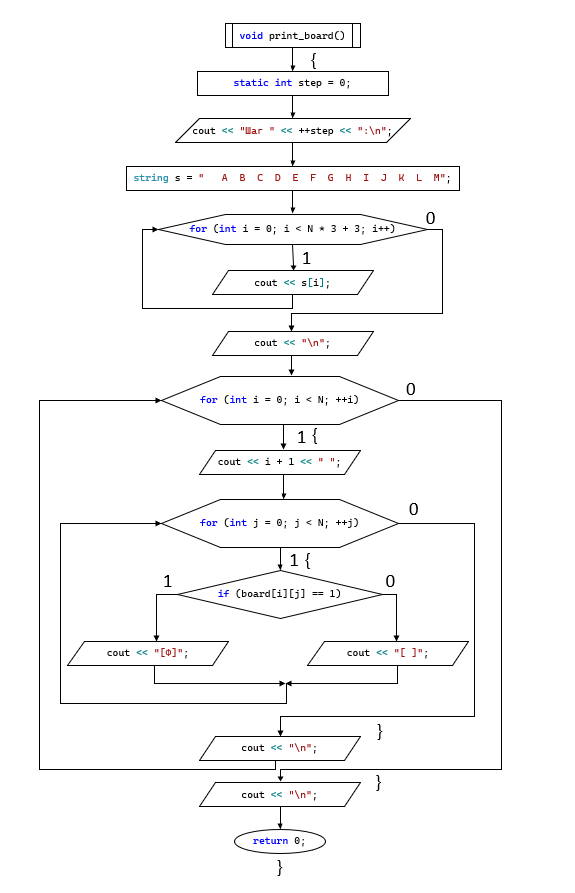


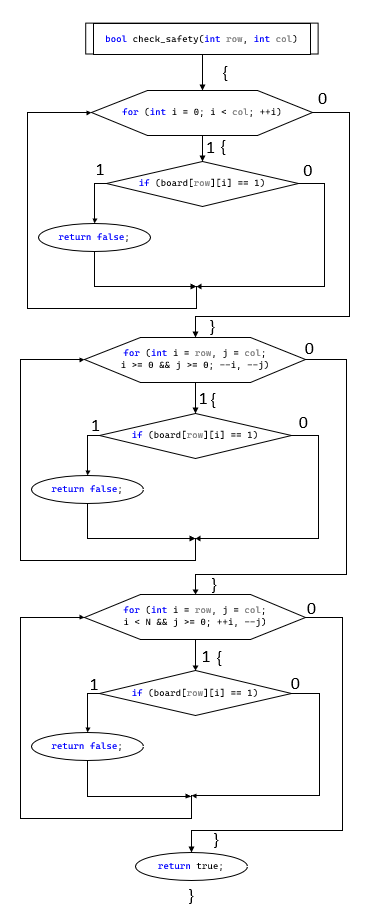


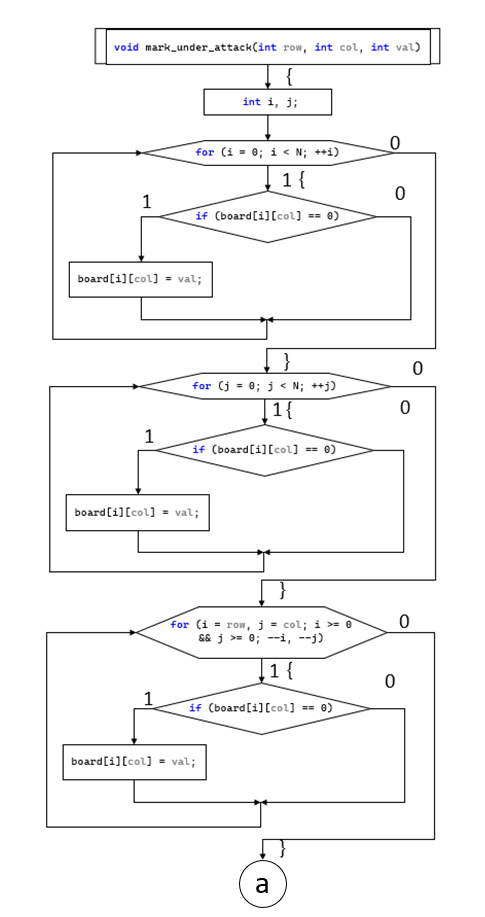


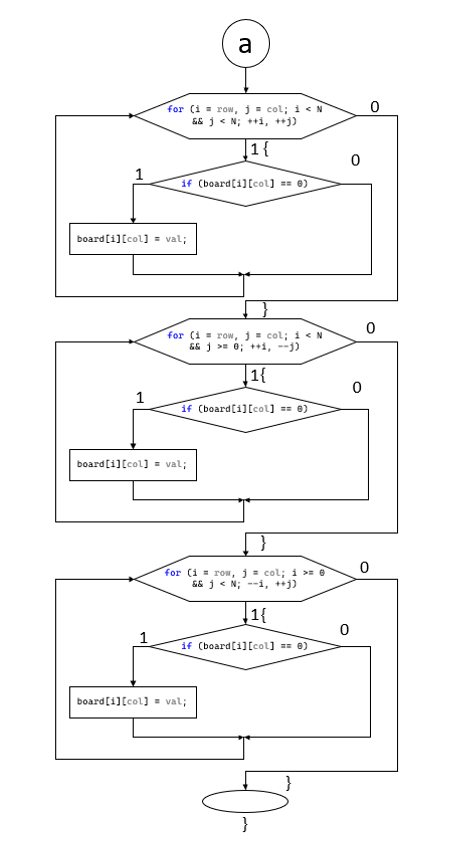
Блок схема программы для поиска всех возможных решений:

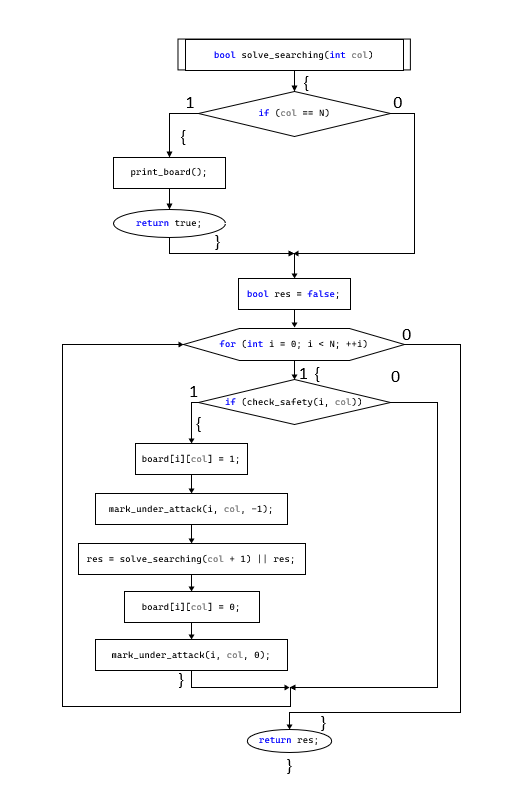












4. Код программы

Код программы для поиска одного любого решения

#include <iostream>

#include <string>

using namespace std;

const int N = 8; //При изменении этого значения мы можем найти решения для поля другого размера

int board[N][N] = { 0 }; // Инициализация шахматной доски (двумерный массив заполненный нулями)

void print\_board(); //Функция для вывода решения на шахмотной доске в консоль

bool check\_safety(int row, int col); //Функция для проверки, находится ли эта клетка под ударом

void mark\_under\_attack(int row, int col, int val); //Функция для отметки клеток, как находящихся под ударом

bool solve\_searching(int col); //Функция для поиска решения

int main() {

setlocale(LC\_ALL, "Rus");

if (!solve\_searching(0)) { //Когда решений задачи нет (2 или 3 ферзя) функция вернет false и мы пройдем по ветвлению

cout << "Нет решений";

}

return 0;

}

void print\_board() {

static int step = 0;

cout << "Шаг " << ++step << ":\n";

string s = " A B C D E F G H I J K L M";

for (int i = 0; i < N \* 3 + 3; i++) { // Подписываем столбцы буквами

cout << s[i];

}

cout << "\n";

for (int i = 0; i < N; ++i) {

cout << i + 1 << " "; // Подписываем строки цифрами

for (int j = 0; j < N; ++j) {

if (board[i][j] == 1) { //Встречая 1 выводим клетку с ферзем

cout << "[Ф]";

}

else { //В остальных случаях выводим пустую клетку

cout << "[ ]";

}

}

cout << "\n";

}

cout << "\n";

}

bool check\_safety(int row, int col) {

//проверяем находится ли эта строка под боем(есть ли в ней ферзи)

for (int i = 0; i < col; ++i) {

if (board[row][i] == 1) {

return false;

}

}

//проверяем находится ли эта строка под боем(есть ли ферзи на этой диагонали слева сверху)

for (int i = row, j = col; i >= 0 && j >= 0; --i, --j) {

if (board[i][j] == 1) {

return false;

}

}

//проверяем находится ли эта строка под боем(есть ли ферзи на этой диагонали слева снизу)

for (int i = row, j = col; i < N && j >= 0; ++i, --j) {

if (board[i][j] == 1) {

return false;

}

}

return true;

}

void mark\_under\_attack(int row, int col, int val) {

int i, j; //Инициализируем заранее что бы не инициализировать затем в 6 циклах

// Помечаем столбец (вертикаль)

for (i = 0; i < N; ++i) {

if (board[i][col] == 0) {

board[i][col] = val;

}

}

// Помечаем строку (горизонталь)

for (j = 0; j < N; ++j) {

if (board[row][j] == 0) {

board[row][j] = val;

}

}

// Помечаем диагонали

// Идем от нашего ферзя влево вниз

for (i = row, j = col; i >= 0 && j >= 0; --i, --j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя вправо вверх

for (i = row, j = col; i < N && j < N; ++i, ++j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя влево вверх

for (i = row, j = col; i < N && j >= 0; ++i, --j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя вправо вниз

for (i = row, j = col; i >= 0 && j < N; --i, ++j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

}

bool solve\_searching(int col) {

if (col == N) {

return true;

}

bool found\_solution = false; // Переменная для отслеживания нахождения решения

for (int i = 0; i < N; ++i) {

if (check\_safety(i, col)) {

board[i][col] = 1; //Ставим ферзя

print\_board(); //Выводим текущую расстановку на доске

mark\_under\_attack(i, col, -1); //Помечаем клетки находящиеся под ударом

found\_solution = solve\_searching(col + 1) || found\_solution;

board[i][col] = 0;

mark\_under\_attack(i, col, 0); // Сбрасываем пометку клеток

if (found\_solution) { // Если нашли решение, завершаем цикл

break;

}

}

}

return found\_solution;

}

Код программы для поиска всех возможных решений

#include <iostream>

#include <string>

using namespace std;

const int N = 9; //При изменении этого значения мы можем найти решения для поля другого размера

int board[N][N] = { 0 }; // Инициализация шахматной доски (двумерный массив заполненный нулями)

void print\_board(); //Функция для вывода решения на шахмотной доске в консоль

bool check\_safety(int row, int col); //Функция для проверки, находится ли эта клетка под ударом

void mark\_under\_attack(int row, int col, int val); //Функция для отметки клеток, как находящихся под ударом

bool solve\_searching(int col); //Функция для поиска решений

int main() {

setlocale(LC\_ALL, "Rus");

if (!solve\_searching(0)) { //Когда решений задачи нет (2 или 3 ферзя) функция вернет false и мы пройдем по ветвлению

cout << "Нет решений";

}

return 0;

}

void print\_board() {

static int solution\_number = 0;

cout << "Решение номер " << ++solution\_number << ":\n";

string s = " A B C D E F G H I J K L M";

for (int i = 0; i < N \* 3 + 3; i++) { // Подписываем столбцы буквами

cout << s[i];

}

cout << "\n";

for (int i = 0; i < N; ++i) {

cout << i + 1 << " "; // Подписываем строки цифрами

for (int j = 0; j < N; ++j) {

if (board[i][j] == 1) { //Встречая 1 выводим клетку с ферзем

cout << "[Ф]";

}

else { //В остальных случаях выводим пустую клетку

cout << "[ ]";

}

}

cout << "\n";

}

cout << "\n";

}

bool check\_safety(int row, int col) {

//проверяем находится ли эта строка под боем(есть ли в ней ферзи)

for (int i = 0; i < col; ++i) {

if (board[row][i] == 1) {

return false;

}

}

//проверяем находится ли эта строка под боем(есть ли ферзи на этой диагонали слева сверху)

for (int i = row, j = col; i >= 0 && j >= 0; --i, --j) {

if (board[i][j] == 1) {

return false;

}

}

//проверяем находится ли эта строка под боем(есть ли ферзи на этой диагонали слева снизу)

for (int i = row, j = col; i < N && j >= 0; ++i, --j) {

if (board[i][j] == 1) {

return false;

}

}

return true;

}

void mark\_under\_attack(int row, int col, int val) {

int i, j; //Инициализируем заранее что бы не инециализировать затем в 6 циклах

// Помечаем столбец (вертикаль)

for (i = 0; i < N; ++i) {

if (board[i][col] == 0) {

board[i][col] = val;

}

}

// Помечаем строку (горизонталь)

for (j = 0; j < N; ++j) {

if (board[row][j] == 0) {

board[row][j] = val;

}

}

// Помечаем диагонали

// Идем от нашего ферзя влево вверх

for (i = row, j = col; i >= 0 && j >= 0; --i, --j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя вправо вниз

for (i = row, j = col; i < N && j < N; ++i, ++j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя влево вверх

for (i = row, j = col; i < N && j >= 0; ++i, --j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

// Идем от нашего ферзя вправо вниз

for (i = row, j = col; i >= 0 && j < N; --i, ++j) {

if (board[i][j] == 0) {

board[i][j] = val;

}

}

}

bool solve\_searching(int col) {

if (col == N) {

print\_board(); //Выводим решение в консоль

return true;

}

bool res = false; // Переменная для отслеживания нахождения решения

for (int i = 0; i < N; ++i) {

if (check\_safety(i, col)) {

board[i][col] = 1; //Ставим ферзя

mark\_under\_attack(i, col, -1); //Помечаем клетки находящиеся под ударом

res = solve\_searching(col + 1) || res;

board[i][col] = 0;

mark\_under\_attack(i, col, 0); // Сбрасываем пометку клеток

}

}

return res;

}

5. Вывод программы

Вывод первой программы (нахождение одного любого решения)

Вывод программы для 5 ферзей:

Вывод программы для 8 ферзей:

Вывод второй программы (нахождение всех возможных решений)

Вывод программы для 1 ферзя:

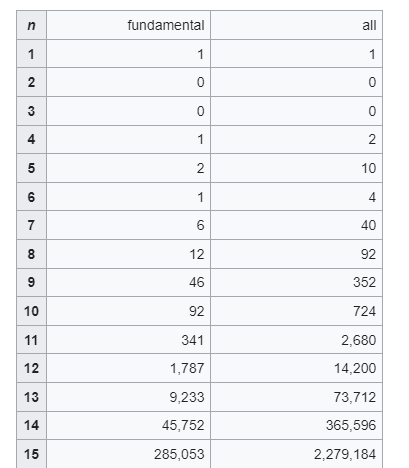
Вывод программы для 2 и 3 ферзей:

Вывод программы для 5 ферзей:

Вывод программы для 8 ферзей:

Вывод программы для 9 ферзей:

Все подсчеты количества решений моей программой совпадают с данными из интернета



(скриншот с сайта Wikipedia, ссылка на страницу: <https://en.wikipedia.org/wiki/Eight_queens_puzzle>)

6. Вывод

7. Github

Ссылка на репозиторий: https://github.com/Dmitriy-Mur/Eight-Queens-Problem/tree/main